TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N 2612 – Elektrotechnika a informatika

Studijní obor: Automatické řízení a inženýrská informatika

Implementace numerického modelu podzemního proudění v objektově orientovaném programovacím jazyce

Implementation of a numerical model of groundwater flow in an object oriented programming language

Diplomová práce

Vedoucí dipl. práce: Ing. Otto Severýn, Ph.D.

2009 Bc. Tomáš Chmel

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména č. 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít svoji diplomovou práci či poskytnout licenci k její využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval sám s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce.

V Liberci 29. května 2009 .....

Abstrakt

Cílem této práce bylo vytvoření programu pro modelování podzemního proudění pomocí objektově orientovaného jazyka. Výchozím bodem byl již existující program, psaný pomocí jazyka C strukturovanou formou.

Na začátku práce je uveden stručný úvod do problému modelování proudění kapalin a objektově orientovanému návrhu programu.

Po vysvětlení obecných pojmů následuje popis konkrétní realizaci programu. Nejprve se zaměříme na činnosti programu, průběh výpočtu a komunikace programu s okolím. Dále je věnována pozornost jednotlivým třídám, navrženým k řešení zadaného problému, především pak jejich metodám.

Konec práce je věnován srovnání nového a původního programu. Srovnání je provedeno jednak pomocí programu GMSH, kterým jsou vypočtené výsledky převedeny do grafické podoby, tak i porovnáním obsahu souborů jako textu.

Klíčová slova: proudění podzemní vody, puklinové prostředí

Abstract

The main goal of this bachelor thesis is to create a programme for groundwater flow modelling using an object-oriented approach. The starting point was the already existing programm written in the programming language C in a structured form.

At the beginning there is a brief introduction to the problem of groundwater flow modelling and concepts of the object-oriented programming.

After the explanation of the basic terminology we will focus on the specific implementation of the programmme. At first, we will focus on the programme functions, calculation phases and programme communication with its environment. Then, we will pay attention to the individual classes created to solve the problem, mainly to their methods.

The end of this thesis concentrates on the comparison of the newly created programme with the original one. This comparison is carried out not only by the programme GMSH, which transfers the calculated results into a graphic form, but also by a comparison of the file content as a text.

Keywords: Groundwater flow, Fractured rock

Obsah

Seznam obrázků

1 Úvod

Cílem práce bylo inovace již existujícího programu flow123d, který slouží k simulaci proudění podzemních vod horninovým prostředím a transportu látek rozpuštěných ve vodě.

Původně byl program psán strukturovaně, čímž je myšleno, že byly vytvořeny globální proměnné ke kterým se přistupovalo pomocí globálních funkcí. Nevýhodou tohoto přístupu se stává jeho složitost, která neúměrně roste s velikostí programu. Program se tak stává hůře udržovatelným. Každá netriviální změna programu se projeví na více místech kódu a je náročné a zdlouhavé tato místa identifikovat.

Naproti tomu objektový přístup, který byl zvolen, slučuje data a metody, které k nim přistupují. Přistup k datům se uskutečňuje přes tyto metody a program se tak dělí do menších celků, které je snažší spravovat.

V první části této práce se věnujeme matematicko-fyzikálnímu popisu problému proudění podzemních vod v horninovém prostředí. Budou vysvětleny základní vztahy nutné k vytvoření programu. Poté budou vysvětleny základní ideje objektového programování a pak se již budeme zabývat programem.

Následně se seznámíme s činností programu a popíšeme vnější datové struktury, které slouží programu jako vstupní data a také jako výstup.

Další část je věnována popisu nových tříd a jejich metod. Nejprve jsou popsány dvě nové parametrické třídy sloužící k vytváření lineárního seznamu. Následuje popis tříd, které odpovídají datovým strukturám v původním programu. Nakonec je popsána třída sloužící k zajištění komunikace programu s okolím.

V poslední kapitole je porovnání původního programu a programu nového z hlediska jejich funkce.

2 Proudění podzemní vody

Součástí této kapitoly bude seznámit čtenáře s použitým modelem pro výpočty proudění v horninovém prostředí.

V horninovém prostředí se můžeme setkat s několika rozdílnými typy objektů, které mají rozdílné charakteristiky.

1. Sedimenty. Typickými zástupci jsou pískovce či vápence. Tyto materiály umožňují, díky své porozitě, proudění kapalin celým objemem.

2. Vyvřelé nebo přeměněné horniny nazývané kristalinikum (např. žuly) . I přes tyto materiály může kapalina protékat přes celý objem, ale je zapotřebí mnohem větších tlaků než v předešlém případě. Dominantním je zde tok po puklinách a dalších tektonických poruchách.

Pukliny v horninovém prostředí můžeme rozdělit podle jejich vlastností.

1. Malé pukliny. Jejich výskyt je velmi hojný, ale dosahují jen malých rozměrů (méně než jeden metr). Modelování každé pukliny je pro současný výkon počítačů nerealizovatelné na území větším než řádově stovky metrů čtverečních v půdoryse a proto se tyto pukliny v modelování nahrazují porézním materiálem ekvivalentních hydraulických vlastností.

2. Významné pukliny. Jejich výskyt je relativně řidší a jejich charakteristiky bývají známé. Tyto pukliny není možné nahradit v modelování porézním materiálem. Ve výpočtech je uvažujeme jako 2D objekty.

3. Křížení velkých puklin. Na těchto místech je možné pozorovat velké toky podzemních vod. Tyto čáry se chovají jako jakési potrubí v podzemním prostředí.

Z výše uvedeného výčtu je možné pro proudění podzemních vod zahrnout 3D porézní bloky, 2D zlomy a 1D pukliny.

2.1 Aproximace proudění

Nejprve zavedeme aproximaci proudění v každém typu nezávisle na ostatních.

Zaveďme oblast Ω_i, kde index $i \in \{1, 2, 3\}$. Ω_1 je množina úseček umístěných ve 3D prostoru, Ω_2 je množina polygonů a Ω_3 množina 3D oblastí. Všechny oblasti jsou podmnožinou třídimenzionálního prostoru. Pro tyto oblasti můžeme definovat potenciál, řídící proudění kapaliny. Pomocí rovnice kontinuity a lineárního Darcyho zákona (který definuje rychlost průtoku kapaliny pevným porézním tělesem) získáme:

$$\mathbf{u}_i = -\mathbf{K}_i \cdot \nabla p_i \text{ na } \Omega_i, \tag{1}$$

$$\nabla \cdot \mathbf{u}_i = q_i \text{ na } \Omega_i, \tag{2}$$

kde u_i je intenzita proudění, p_i je tlaková výška, $\mathbf{K}_i$ je tenzor hydraulické vodivosti a q_i je funkce popisující hustotu zdrojů kapaliny.

Při řešení uvažujeme tři možné okrajové podmínky - Dirichetovu, Neumanovu a Newtnovu, vyjádřené vztahy:

$$p_i = p_{iD} \text{ na } \partial\Omega_{iD}, \tag{3}$$

$$\mathbf{u}_i \cdot \mathbf{n}_i = u_{iN} \text{ na } \partial\Omega_{iN}, \tag{4}$$

$$\mathbf{u}_i \cdot \mathbf{n}_i - \sigma(p_i - p_{iD}) = u_{iN} \text{ na } \partial\Omega_{iW}. \tag{5}$$

Diskretizací získáme soustavu lineárních algebraických rovnic. Pro tento případ byla použita smíšená-hybridní formulace MKP (více o diskretizaci v [5]). Výsledná soustava má tvar:

$$\mathbb{S}_i \mathbf{x}_i = \mathbf{r}_i, \tag{6}$$

kde $x_i = [\mathbf{u}_i, \mathbf{p}_i, \lambda_i]^T$, $r_i = [\mathbf{r}_{i1}, \mathbf{r}_{i2}, \mathbf{r}_{i3}]^T$ a matice

$$\mathbb{S}_i = \begin{pmatrix} \mathbb{A}_i & \mathbb{B}_i & \mathbb{C}_i \\ \mathbb{B}_i^T & & \\ \mathbb{C}_i^T & & \mathbb{F}_i \end{pmatrix} \tag{7}$$

Symbol λ_i udává tlakovou výšku ve středu elementů.

2.2 Sloučení 1D, 2D a 3D výpočtu

Nyní přikročíme k propojení jednotlivých částí výpočtu problému. Začneme tím, že vytvoříme jeden velký systém obsahující všechny části

$$\mathbb{S}\mathbf{x} = \mathbf{r}, \tag{8}$$

kde $x_i = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3]^T$, $r_i = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]^T$ a matice

$$\mathbb{S} = \begin{pmatrix} \mathbb{S}_1 & & \\ & \mathbb{S}_2 & \\ & & \mathbb{S}_3 \end{pmatrix}. \tag{9}$$

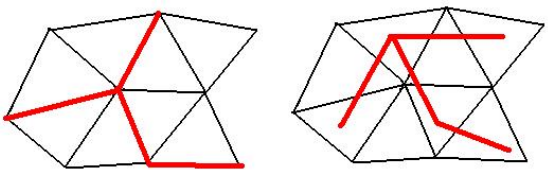
2.2.1 Konformní a nekonformní propojení elementů

Ve výpočtu můžeme uvažovat dva druhy propojení jednotlivých elementů různé dimenze.

Prvním druhem je konformní propojení. V tomto případě prvky nižší dimenze leží pouze na hranách či stěnách prvků vyšší dimenze. Tento způsob je velmi náročný na vytváření sítě při složitějších tvarech zkoumaných oblastí.

Druhý způsob, nazývaný nekonformní propojení, nepožaduje žádné speciální umístění prvků nižší dimenze vzhledem k ostatním prvkům.

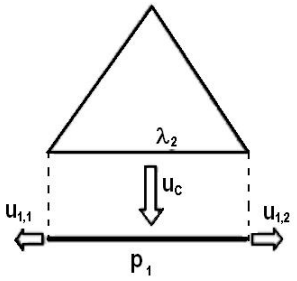
Názorná ukázka jednotlivých možností propojení je znázorněna na obrázku 1.



Obrázek 1: Konformní a nekonformní propojení prvků s různou dimenzí. Převzato z [2].

2.2.2 Výměna látek přes konformní propojení elementů

Pro odvození je použito propojení 1D a 2D elementů, jak je nakresleno na obrázku 2. Propojení o řád vyšších elementů se provede analogicky.



Obrázek 2: Schéma látkové výměny přes konformní propojení. Elementy jsou pro názornost nakresleny posunuté ve směru čárkovaných čar. Převzato z [2]

Propojení se chová jako nehomogení okrajová podmínka pro element s vyšší dimenzí a jako zdroj pro element s nižší dimenzí.

$$u_{11} + u_{12} = u_c \tag{10}$$

Při určení toku mezi jednotlivými elementy se předpokládá, že tok je přímo úměrný velikosti gradientu tlaků mezi jednotlivými elementy

$$-u_C = \sigma_C(\lambda_2 - p_1), \tag{11}$$

kde λ_2 je tlak na straně 2D elementu, p_1 je tlak uprostřed 1D elementu a σ_C je koeficient přestupu.

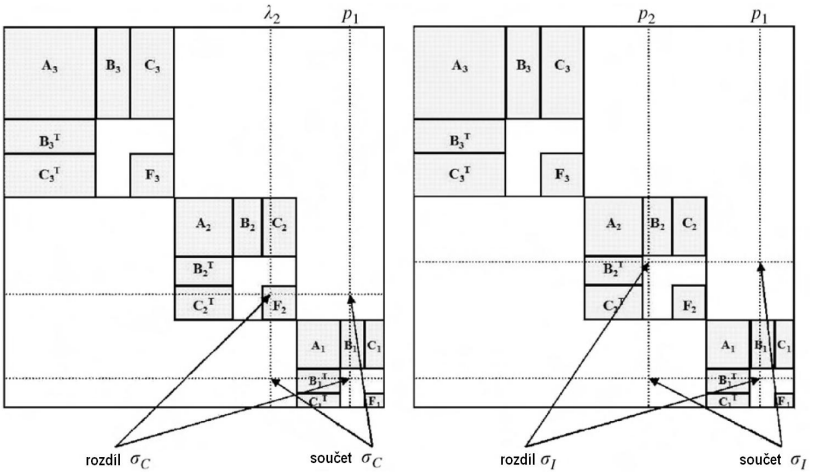
Zapsaní podmínek dostaneme vztahy

$$-u_C - \sigma_C\lambda_2 - \sigma_C p_1 = 0 \tag{12}$$

$$\sigma_C\lambda_2 - u_{1,1} - u_{1,2} - \sigma_C p_1 = 0. \tag{13}$$

Porovnáním původních rovnic a odpovídajících rovnic nových můžeme určit potřebné změny matice $\mathbb{S}$. Tyto změny jsou znázorněny v levé části obrázku 3.

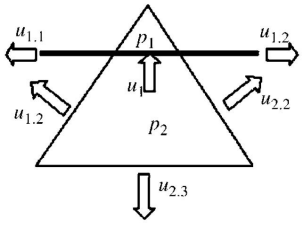
Propojení nelze realizovat pro 1D elementy a 3D elementy.



Obrázek 3: Zavedení konformního (vlevo) a nekonformního (vpravo) propojení elementů v globální matici. Převzato z [2]

2.2.3 Výměna látek přes nekonformní propojení elementů

Nyní si odvodíme proudění mezi dvěma elementy s nekonformní vazbou. Stejně jako v předešlém případě se zaměříme na vazbu mezi 1D a 2D elementy. Vztahy pro ostatní propojení jsou odvozeny analogicky. Schéma možné vazby je na obrázku 4.



Obrázek 4: Schéma látkové výměny přes nekonformní propojení 1D a 2D elementů. Převzato z [2]

Tok mezi elementy je možné popsat vztahem

$$u_1 = \sigma_I(p_2 - p_1), \tag{14}$$

kde p_1 je tlak v těžišti 1D elementu, p_2 je tlak v těžišti 2D elementu a ko-

eficient σ_I je závislý na vzdálenosti středů a velikosti křížení jednotlivých elementů.

Rovnice zachování hmoty pro 2D element je

$$-u_{2,1} - u_{2,2} - u_{2,3} - u_I = 0, \tag{15}$$

podobně je udán i vztah pro 1D element

$$-u_{1,1} - u_{1,2} + u_I = 0. \tag{16}$$

Spojením těchto rovnic získáme výsledné vztahy

$$-u_{2,1} - u_{2,2} - u_{2,3} - \sigma_I p_2 + \sigma_I p_1 = 0 \tag{17}$$

$$-u_{1,1} - u_{1,2} + \sigma_I p_2 - \sigma_I p_1 = 0. \tag{18}$$

Je vidět, že propojení jednotlivých elementů lze zajistit přičtením nebo odečtením σ_I v matici $\mathbb{S}$ jak názorně ukazuje pravá část obrázku 3.

3 Objektově orientované programování

K programu lze z hlediska vývoje přistupovat několika různými způsoby. V této kapitole se zaměříme na základní vlastnosti objektově orientovaného přístupu. Více o objektovém programování je v [1].

Základem objektového přístupu je modelování jednotlivých prvků řešené úlohy, jak dat, tak i souvisejících funkcí do jedné entity, nazývané objekt. Objekt je schopen měnit svůj stav a poskytuje operace, které jej mohou změnit nebo informovat o jeho stavu.

Abstrakce objektu, která popisuje jeho tvar v rámci programovacího jazyka se říká třída. Třída obsahuje instrukce jak daný objekt vytvořit, jaké má atributy (datové proměnné) a metody (procedury a funkce). Objektem je nazývána konkrétní instance této třídy.

3.1 Dědičnost

Při vytváření objektů se můžeme rozhodnout, že nový objekt bude mít vlastnosti již existujícího objektu, jelikož obě reálné předlohy pro daný model jsou podobné, ale nový objekt bude mít několik vlastností navíc. Můžeme vytvořit novou třídu a dané metody a atributy do ní přepsat. V tomto případě však bude problém při další změně, jelikož bychom museli zdrojový kód upravovat na dvou místech. Proto objektové programování umožňuje vytvářet potomky třídy. Tento potomek dědí všechny atributy a metody předchozí třídy, které budou stále zapsány na jednom místě v programu a v případě úpravy je stačí přepsat pouze tam. Odvozená třída přitom stále funguje jako objekt původní třídy a na místě, kde je očekávána původní třída se může vyskytovat i její potomek.

3.2 Polymorfismus

Je vlastnost objektového programování spojená s dědičností. Umožňuje překrytí metody bázové třídy metodou odvozené třídy.

Ještě dále se dostaneme pomocí virtuálních metod. Ty jsou propojeny dynamicky za běhu programu. Lze tedy pomocí ukazatele na bázovou třídu ukazovat na odvozenou třídu a při volání virtuální metody se zavolá metoda deklarovaná v odvozené třídě.

3.3 Skládání objektů

Dalším využitím objektů je možnost rozložit program na jednotlivé části, vykonávající jednotlivé úlohy. Tyto části mají definované vstupy a výstupy, ale postup řešení, nebo-li to co je uvnitř, je z hlediska jejich použití nepodstatné.

Jeden objekt může vlastnit odkazy na jiné objekty a využívat tak jejich metody, bez nutnosti znát konkrétní implementaci. Při údržbě programu pak může být například ve vnořeném objektu změněn způsob uložení dat. Pokud si objekt zachová stejné rozhraní není nutné upravovat další části programu.

3.4 Výhody a nevýhody objektového přístupu

Objektově orientovaný program bývá přehlednější, srozumitelnější a proto je snažší ho spravovat a inovovat. Před objektovým programováním se používal procedurální přístup, který spočíval ve vytvoření globálních dat, ke kterým se přistupovalo pomocí globálních funkcí. Zde plyne hlavní nevýhoda z globálního přístupu k datům. Přestože je vývoj takovýchto programů z počátku rychlý, s rostoucí velikostí a složitostí, a tedy i velikostí zdrojových kódů se zvyšuje nepřehlednost. Při změně programu nastává nutnost upravovat program na více místech.

Při vývoji objektového programu je však kladen důraz na zapouzdření, které umožňuje přístup k datům jen metodám dané třídy a program je tak možné udržovat přehledný a tedy je snažší provádět údržbu či vylepšování programu.

Nevýhodou tohoto přístupu však bývá větší časová náročnost, jelikož k přístupu k datům se používají funkce a program se prodlužuje o náklady

spojené s voláním funkcí. To se dá však minimalizovat použitím řádkových funkcí, kdy překladač místo vytvoření volání funkce vloží příkazy z funkce do místa volání. Tím se ušetří čas na volání funkce, ale zvětší se velikost programu, jelikož daný obsah funkce je vložen tolikrát, kolikrát je daná funkce volaná.

4 Popis programu flow123d

V této kapitole se budeme zabývat popisem programu a jeho vnějšími datovými strukturami.

Nejprve představíme základní koncepci programu flow123d, dále pak popíšeme jeho vstupní a výstupní formáty.

4.1 Režimy programu flow123d

Celý program byl realizován jako jedna aplikace, umožňující použití tří módů činností:

1. Režim generování souboru sousedností
2. Režim konverze vstupních souborů
3. Režim výpočtu

Pro spuštění programu je nutné zadat dva parametry v příkazové řádce. Prvním parametrem je druh činnosti, kterou má program vykonat a druhým je řídící soubor.

4.1.1 Generování souboru sousednosti

Během tohoto režimu dojde k vytvoření souboru sousednosti, který obsahuje informaci o topologii sítě. Jeho vytvoření je časově náročné (kvadratická náročnost na čase) a proto je výhodné tuto činnost oddělit.

4.1.2 Konverze vstupních souborů

Tento mód složí jako pomoc při tvorbě modelu. Během běhu programu jsou načteny vstupní soubory, které jsou převedeny do formátu POS. Ten je možné zobrazit pomocí programu GMSH.

4.1.3 Výpočet

Po načtení vstupních souborů je provedena vlastní simulace proudění a transportu.

4.2 Vnější datové struktury

Ve všech případech jde o textové soubory které jsou postupně, v závislosti na druhu činnosti programu, načteny po spuštění.

4.2.1 Řídící soubor

Řídící soubor slouží konfiguraci činnosti programu. Jeho formát je shodný s formátem konfiguračních INI souborů systému MS Windows.

Soubor je složen ze sekcí, ve kterých obsahuje řádky typu klíč = hodnota. Předností tohoto formátu je snadná editace a možnost přidávání nových sekcí při rozšíření programu.

4.2.2 Soubor sítě

Slouží k prostorovému popisu sítě. Pro tento soubor byl zvolen formát systému GMSH. Jde o velmi obecný formát a program flow123d využívá pouze několik možností tohoto formátu. Podrobnou specifikaci tohoto formátu je možné najít v [3]. Standardně se pro tento soubor používá přípona .MSH. Ostatní vstupní soubory jsou podobného stylu a proto se mu budeme věnovat trochu více.

Jde o textový soubor, rozčleněný na sekce. První sekce často slouží k zaznamenání informace o typu souboru a jeho verzi. Ostatní pak obsahují data. Informace je organizována po řádcích. Každý řádek nese informaci o jednom objektu. Objekt je definován jednoznačně svým identifikačním číslem v rámci typu. Identifikační číslo patří do nezáporných, celých čísel. Pomocí něj se na tento objekt mohou odkazovat ostatní objekty. Na uspořádaní objektů v rámci souboru není kladena žádná podmínka.

Kromě identifikačních čísel mohou jednotlivé objekty obsahovat *tagy*. Jde o množinu celých nezáporných čísel. Jejich počet a význam je volitelný a záleží na uživateli jak je použije. Je možné mít různý počet tagů i v rámci jednoho typu. Program flow123d definuje některé tagy jako povinné a přiděluje jim konkrétní vlastnosti.

To bylo k obecnému formátu a nyní se podíváme konkrétně na soubor sítě. Ten obsahuje dvě datové sekce. První je určena pro uzly a druhá pro elementy. Obsahem uzlu je jeho pozice pomocí kartézských souřadnic. Element obsahuje informaci o svém typu, identifikační čísla uzlů ze kterých je tvořen a své tagy. První tag je definován jako odkaz na materiál, ze kterého se element skládá.

4.2.3 Soubor materiálových vlastností

Soubor je označován příponou .MTR. Obsahuje informace o jednotlivých materiálech v oblasti, kde simulace probíhá. Soubor je rozdělen do sekcí, kde každá sekce popisuje jednu fyzikální veličinu, určenou jménem sekce. V sekci jsou dále definovány jednotlivé materiály a jim je přiřazená konkrétní hodnota.

4.2.4 Soubor okrajových podmínek

Daný model proudění umožňuje definovat okrajové podmínky Dirichletova, Neumannova a Newtonova typu. Implicitně je na každé hranici uvažována homogenní podmínka Neumanova typu. Ta je předpokládána všude, kde není uvedena explicitně jiná podmínka.

Všechny okrajové podmínky v souboru mají definovaný svůj typ. Na něm závisí počet a význam dalších uvedených hodnot. Další informace je o objektu, kde je podmínka zavedena (uzel či stěna) a tento objekt je identifikován. Nakonec definice okrajové podmínky jsou uvedeny tagy, kde první určuje skupinu. Model proudění pak sčítá objem kapaliny prošlé skrz hranici pro jednotlivé skupiny.

4.2.5 Soubor zdrojů kapaliny

Soubor označovaný příponou .SRC, slouží k určení zdrojů uvnitř oblasti. Zdroje jsou spojeny s konkrétním elementem a určují intenzitu vtlačení či čerpání objemu za čas. Zavedená konvence je, kladná hodnota pro vtlačení a záporná pro čerpání.

Tento soubor je volitelný a pokud není uveden v řídícím souboru, výpočet bude probíhat bez uvažování vnitřních zdrojů.

4.2.6 Soubor sousednosti elementů sítě

Tento soubor je odlišný od ostatních. Především je tento soubor generován ve speciálním běhu programu flow123d. Druhá odlišnost spočívá v tom, že tento soubor není pro běh programu nutný, jelikož všechny informace jsou již obsaženy v souboru sítě.

Zpracování a zjištění sousednosti je časově náročná úloha. Často ji stačí zjistit jen jednou pro různé výpočty, kde se mění materiálové či okrajové podmínky na stejné síti. Vytvořením tohoto souboru tedy lze výrazně snížit čas výpočtu.

Soubor má jednu datovou sekci, kde na každém řádku je definovaná množina propojených stěn a/nebo elementů. Přípona souboru je .NGH.

4.2.7 Soubor okrajových podmínek transportu

Soubor označovaný příponou .TRB zavádí hodnoty počátečních koncentrací na hranici oblasti. Tyto hodnoty jsou spojeny s okrajovými podmínkami proudění. Podobně jako okrajové podmínky proudění i zde je implicitně předepsaná podmínka a to nulová. Jinak řečeno, pokud není předepsaná jiná podmínka, uvažuje se nulová koncentrace látek ve vtékající kapalině. Pokud je podmínka předepsaná v místě kde kapalina opouští oblast je tato podmínka ignorována.

4.2.8 Soubor počátečních podmínek

Soubor s příponou .ICD je zpravidla generován výpočtem. Používá se při výpočtu neustáleného proudění a transportu. Soubor obsahuje proměnný počet sekcí, pro několik časových okamžiků. V každé sekci jsou vypsány hodnoty tlaků v těžišti elementů a stěn a hodnoty objemových toků přes stěny elementu.

4.2.9 Soubor počátečního rozložení koncentrace

Tento soubor obsahuje počáteční rozložení koncetrací. Informace je zapsána pro každý element sítě a udává hodnoty koncentrací pro začátek simulace. Tento soubor má příponu .CON.

4.2.10 Výstupní soubory

Program flow123d generuje tři druhy výstupních souborů. První soubor s příponou .POS slouží k vizualizaci výsledků. Druhé dva je pak možné použít jako vstupní hodnoty pro další simulaci.

POS-soubory jsou určené k vizualizaci pomocí programu GMSH. Výsledky jsou uloženy pomocí zadaných bodů v prostoru a vypočtené hodnoty jsou zadány formou souřadnic. Tento formát není vhodný pro další použití ve výpočtu, jelikož se ztratila vazba na síť.

Proto byly zavedeny další dva soubory které jsou formálně totožné se soubory počátečních podmínek (ICD-soubor) a počátečního rozložení koncentrace (CON-soubor).

5 Vnitřní datové třídy

V následující části budou popsány jednotlivé třídy v programu. Předtím než se začneme věnovat třídám odpovídajícím strukturám v původním programu, budeme se věnovat parametrizovaným typům neboli šablonám. Poté se podíváme na dvě šablony použité k vytvoření propojených seznamů.

5.1 Šablony

Šablony jsou zabudované nástroje jazyka C++ a umožňují vytvářet parametrizované typy. Tyto typy mohou měnit svoje chování dle parametrů zadaných při jejich vytvoření. Jejich použití je výhodné zejména tam, kde se opakuje téměř shodný kód. Jedním z těchto případů je vytváření seznamu tříd, jelikož se z velké části mění jen typ ukládané třídy.

Ve vytvářeném programu bylo použito dvojice šablon k vytvoření obousměrného lineárního seznamu. První parametrizovaný typ slouží k uložení jedné třídy. Druhý pak k obsluze metod celého seznamu, jako například přidávání nových prvků či jejich vyhledávání.

5.1.1 Šablona CList_node

Je základním stavebním prvkem seznamu. Jako parametr obsahuje ukazatel na ukládanou třídu. Ve výpisu 5 je uvedena její deklarace.

Na prvním řádku je uvedena definice parametru použitého v této třídě. Na řádcích čtyři až šest jsou definovaná data objektu. Ve všech třech případech jde o ukazatele. První dva jsou určeny k vytváření seznamu. Poslední pak ukazuje na ukládaná data.

Na ostatních řádcích jsou definované metody umožňující nastavování a přístup k těmto datům.

CList_node() je konstruktor třídy. Je volán překladačem vždy, když je vytvořena nová instance třídy. Slouží k nastavení dat na počáteční hodnoty. V tomto případě nastaví všechny ukazatele na hodnotu *NULL*.

Obrázek 5: Výpis třídy CList_node

```
1 : template <class T_class>
2 :class CList_node{
3 : private:
4 :   CList_node  * Next;
5 :   CList_node  * Prev;
6 :   T_class     * Data;
7 : public:
8 :                CList_node();
9:                CList_node(const CList_node & );
10:               ~CList_node();
11:   CList_node  *next();
12:   CList_node  *prev();
13:   T_class     *data();
14:   void         set(T_class* );
15:   void         next(CList_node *);
16:   void         prev(CList_node *);
17:   void         free();
18:};
```

CList_node(const CList_node &) je kopírovací konstruktor. Slouží k nastavení chování při kopírování instancí tříd.

CList_node() je destruktor. Je volán kompilátorem vždy když dochází k likvidaci instance. Datová instance uložená v tomto uzlu není rušena.

***next()** vrací ukazatel na další prvek v seznamu.

***prev()** vrací ukazatel na předchozí prvek v seznamu.

***data()** vrací ukazatel na parametrizovaný typ.

set(T_class*) slouží k nastavení ukazatele na objekt, který má být vložen do seznamu.

next(CList_node *) je použit při vytváření seznamu, kdy se nastaví následující prvek seznamu. Pokud je tento prvek posledním, nenastavuje se nic a implicitně zůstává hodnota *NULL*.

prev(CList_node *) je použit při vytváření seznamu, kdy se nastaví předchozí objekt.

free() slouží k destrukci objektu který je uložen v ukazateli *Data*. Po jeho likvidaci je ukazateli přiřazena hodnota *NULL*.

5.1.2 Šablona CList

Tato šablona byla navržena jako správce seznamu. Parametrem je opět ukládaná třída. Kromě vytvoření a správy seznamu umožňuje tato šablona přístup k jednotlivým prvkům pomocí jejich identifikačního čísla. Identifikační číslo musí být již součástí ukládané třídy a musí být typu *int*.

Jelikož se jedná o šablonu musí deklarace začínat definováním parametru. Na dalších řádcích jsou pak definovány jednotlivá data. Tato data se dají rozdělit do dvou skupin. Tou první jsou data nutná k funkčnosti seznamu. Těmi jsou jen ukazatelé na první a poslední prvek seznamu.

Ostatní data slouží pouze k urychlení nebo zjednodušení činnosti. Sem patří například ukazatel na aktuální prvek, který usnadňuje procházení celého seznamu.

V deklaraci je opět definován konstruktor, destruktor a metoda k uvolnění dat v celém seznamu (*free()*). Dále pak odkazy na první a poslední prvek seznamu a jejich indexy. Další metody se budou popsány podrobněji:

add(T_class *, int =0) slouží k přidání prvku do seznamu. Nový prvek je přidán na konec seznamu.

init(int) vytváří nový seznam prvků. Předávaným argumentem je počet členů seznamu. Při vytváření jsou vytvořeny i jednotlivé prvky.

init_empty(int) vytváří nový seznam prvků. Předávaným argumentem je počet členů seznamu. Při použití této metody je vytvořen pouze prázdný

seznam. Tím je myšleno vytvoření a propojení jednotlivých uzlů seznamu, které ovšem neobsahují žádná data.

set(int,T_class *) nastaví uzlu na dané pozici zadanou třídu.

make_hash() vytvoří pole ukazatelů na parametry dané instance. Pole je uložené v datové položce *Hash*. Počet ukazatelů je dán maximálním identifikačním číslem v již existujícím seznamu. Při vytváření pole jsou všechny ukazatelé nastaveny na hodnou *NULL*. Po vytvoření pole se projde celý seznam a každému prvku se podle jeho identifikačního čísla přiřadí pozice ve vytvořeném poli. Nakonec je nastaven příznak *Hash_init*. Tento příznak je vynulován při změně struktury seznamu (přidání nového prvku).

operator [ ](int) - tato definice deklaruje operátor, pomocí kterého lze přistupovat k jednotlivým prvkům pole přes index. Před navrácením ukazatele na daný prvek je ověřeno, že daný prvek může existovat (index není menší než nula a větší než maximální počet prvků).

Po této kontrole se určí, odkud je nejlepší procházet seznam k nejrychlejšímu nalezení dané pozice. Nejprve je zkontrolováno, zda pozice hledaného prvku není o jednu větší než je pozice aktuální. To je prováděno kvůli maximálnímu zrychlení při procházení seznamu prvek po prvku. Pokud to není splněno zjistí se který ze tří ukazatelů je nejblíž (první, aktuální nebo poslední) a od něj se začne procházet seznam. Ukazatel na aktuální prvek je pak přesunut na hledaný prvek.

operator ()(int) - pomocí této definice je vytvořen operátor umožňující přístup k jednotlivým položkám pomocí jejich identifikačního čísla. Tato metoda je použitelná pouze až po vytvoření pole. Před navrácením ukazatele se kontroluje, jestli daný prvek může existovat a pak je vrácena hodnota na pozici odpovídající zadané hodnotě. Další kontrola v tomto operátoru neprobíhá takže je možné, že vrácená hodnota bude *NULL* (prvek daného čísla není v tomto seznamu).

next() vrací ukazatel na prvek nacházející se za aktuálním prvkem v seznamu. Zároveň dojde k posunu aktuálního prvku. Díky tomuto lze rychleji přistupovat k prvkům nacházejícím se v seznamu za sebou. Nebezpečí použití spočívá ale v tom, že aktuální prvek je také změněn při vyhledávání pomocí operátoru [ ], nebo při vložení nových dat. Pokud je aktuální prvek na konci seznamu je vrácena hodnota *NULL*.

prev() pracuje podobně jako předchozí metoda, jen vrací ukazatel na předcházející prvek.

Obrázek 6: Výpis třídy CList

```
1 :template <class T_class>
2 :class CList{
3 : private:
4 :  CList_node<T_class> * First;
5 :  CList_node<T_class> * Last;
6 :  CList_node<T_class> * Aktual;
7 :  T_class             **Hash;
8 :  long int              Max_id;
9 :  bool                  Hash_init;
10:  unsigned int          Flag;
11:  long int              N_list_node;
12:  long int              Aktual_poz;
13:
14:  void                  add_empty();
15:public:
16:             CList();
17:            ~CList();
18: void        free();
19: void        add(T_class *, int =0 );
20: void        init(int);
21: void        init_empty(int);
22: void        set(int,T_class *);
23: void        make_hash();
24: T_class   * operator [](int poz);
25: T_class   * operator ()(int poz_id);
26: T_class   * next()  ;
27: T_class   * prev()  ;
28: T_class   * first() ;
29: T_class   * last()  ;
30: int         first_index() const;
31: int         last_index() const;
32: int         max_id() const;
33: bool        data_init()const;
34:};
```

5.2 Třídy

V programu flow123d v popisované verzi jsou implementovány následující třídy:

- Úloha (CProblem)
- Síť (CMesh)
- Uzel (CNode)
- ELement (CElement)
- Stěna (CSide)
- Hrana (CEdge)
- Materiál (CMaterial)
- Okrajová podmínka (CBoundary)
- Počáteční podmínka (CInitial)
- Okrajová podmínka transportu (CTransport_bcd)
- Počáteční rozložení koncentrace (CConcentration)
- Zdroj (CSource)
- Soused (CNeighbour)
- Matice (CMatrix)
- Chyba (CError)
- Systém (CSystem)

Většina těchto tříd má přímou vazbu na vstupní soubory (vnější datové struktury), ze kterých jsou hodnoty kopírovány do příslušných datových členů. Kromě Úlohy, Sítě, Matice a Systému existují výše uvedené třídy v mnoha

instancích. Pro jejich uložení byla použita výše popsaná šablona (mimo třídu Chyby, jejíž instance jsou uloženy pomocí pole a zadány jsou přímo ve zdrojovém kódu).

5.2.1 CProblem

Jde o datovou třídu která v programu flow123d má jedinou instanci. Třída je vytvořená ještě před spuštěním hlavní procedury, což zaručuje, že při náhlém ukončení programu, ke kterému dochází při programovém zjištění chyby, dojde po vypsání údajů k zavolání destruktoru této třídy. Obsahuje všechny údaje z INI-souboru, podle kterých se pak odvíjí celý běh programu.

Deklarace této třídy je značně obsáhlá a proto zde nebude uveden kompletní výpis (ten je uveden v hlavičkovém souboru *problem.hpp*), ale zmíním se jen o několika podstatných metodách této třídy.

get_params slouží ke kontrole vstupních argumentů předaných v příkazové řádce. Pokud jsou zadány neadekvátně, program vypíše očekávaný vstup a po stisknutí klávesy ENTER se ukončí.

make slouží k načtení řídícího souboru. Celá metoda je rozdělena do dvou části řešených pomocí privátních metod:

1. Metoda *read_ini_file* určená k načtení řídícího souboru.
2. Metoda *check_ini_values* slouží ke kontrole, zda byly zadány všechny potřebné vstupní soubory a jestli základní konstanty jsou zadány ve správném rozsahu možných hodnot.

make_mesh slouží k načtení vstupních souborů a vytvoření seznamů ostatních tříd. To, které seznamy budou vytvořeny, závisí na typu úlohy. Dále pak jsou vytvořená indexová pole (pomocí parametrizované třídy Clist popsané v 5.1.2).

topology slouží k propojení jednotlivých tříd pomocí ukazatelů místo identifikačních čísel, která byla zadána ve vstupních souborech. V této

metodě je důležité zachovat správné pořadí příkazů (jde ve většině případech o metody třídy CMesh) jelikož některé metody již předpokládají zadané ukazatele pomocí předešlých metod.

preprocess slouží k nastavení počátečních hodnot do jednotlivých elemementů.

process větví program podle zadané úlohy. Pro každou je vytvořena příslušná metoda počítající daný úkol.

posprocess převádí vypočtené výsledky z matice do daných tříd a počítá transport koncentrací (jestliže to je součástí zadání). Na rozdíl od předchozích metod je tato metoda soukromá.

output je poslední uvedenou metodou. Slouží k převedení hodnot uložených v jednotlivých třídách do výstupních souborů.

5.2.2 CMesh

Stejně jako předešlá třída i tato má během programu jen jednu instanci. Třída slouží ke skladování ostatních instancí tříd, především v podobě seznamu realizovaného pomocí výše zmíněné šablony CList. Pro každý druh existuje jedna instance této parametrizované třídy.

Metody této třídy lze rozdělit do několik skupin podle činnosti, kterou vykonávají:

1. Základní metody. Slouží k nastavení či k dotazu na konkrétní data v třídě. Jde především o nastavení jmen souborů, ze kterých jsou pak načítána data. Mezi dotazy pak patří především počet prvků nějaké instance (elementů, uzlů ...). Tyto metody jsou většinou velmi jednoduché.

2. Metody pro načtení dat. Slouží pro načtení dat ze souborů. Při načítání je nejprve zjištěn a poté vytvořen potřebný počet instancí daného typu.

Poté se každé instanci předá jeden řádek ze vstupního souboru k nastavení dat.

Do této skupiny patří i metody, které nemají vstupní soubory, jelikož potřebné informace jsou již načteny. Jde o metody pro vytvoření seznamu Stěn a Hran (více o tom co je myšleno stěnou a hranou je pojednáno v 5.2.6 a 5.2.5). Všechny metody z této skupiny jsou volány při vytváření sítě již zmiňovanou metodou *make_mesh* předešlé třídy.

3. Metody pro vytvoření indexového pole. Tyto metody jsou opět velmi jednoduché, jelikož se vždy jedná jen o zavolání metody parametrizované třídy CList.

4. Metody topologie. Metody této skupiny slouží k propojení instancí pomocí ukazatelů odpovídajících identifikačním číslům. V průběhu těchto metod se nejčastěji objeví chyba návrhu sítě. Tím je myšleno například identifikační číslo odkazující na neexistující materiál.

5. Metody výpisu. Jde o metody konvergující vnitřně uložená data do výstupních formátů. Jsou dělena podle třídy, která se má vypsat a většinou jde o procházení celého seznamu a postupné volání vypisující funkce pro danou třídu.

6. Metody výpočtu. Jde o pomocné výpočty sloužící k vytvoření potřebných údajů pro dosazení do hlavní matice a následně pro vyčtení dat ze získaných výsledků a jejich přiřazení do vnitřních datových struktur.

7. Metody pro tvorbu sousednosti. Poslední skupinou jsou metody pro zjištění sousednosti jednotlivých prvků.

5.2.3 CNode

Třída určená k nesení informace o poloze v prostoru. Kromě své polohy obsahuje informaci o velikosti koncentrace v daném místě, skalární veličině v tomto i předešlém kroku výpočtu. Skalární veličinou je v modelu tlak.

Tato třída muže být v jiné formulaci mnohem důležitější.

Kromě těchto veličin určených přímo k výpočtu jsou součástí třídy dvě dynamicky alokovaná pole, obsahující ukazatele na elementy a stěny ke kterým tato třída náleží.

Metody této třídy umožňují přístup k datovým členům. Pokud se jedná o dynamicky alokovaná data, je provedena kontrola, zda je daný index platný.

5.2.4 CElement

Je největší a nejdůležitější třídou z hlediska výpočtu. V této třídě se před výpočtem shromažďuje většina potřebných dat. Navíc obsahuje značný počet odkazů na ostatní objekty.

Metody této třídy se dají rozdělit do několika skupin:

1. Metody určené ke čtení stavu třídy. Poskytují pouze jednosměrný přístup k datům a to čtení. Pokud jde o data, která jsou ukládaná do pole, je prováděn test, jestli dotaz na hodnotu není mimo toto pole.

2. Metody určené k nastavení dat. Tyto metody ukládají zadanou hodnotu. Ve všech případech začínají slovem *set*. Pokud se jedná o hodnoty ukládané do pole je provedena kontrola, zda zadaná pozice není mimo vytvořené pole. Spolu s předchozí skupinou tvoří nejpočetnější metody této třídy.

3. Výpočtové metody. Slouží k nastavení proměnných z již uložených dat nebo pomocí odkazů na jiné objekty. Jde například o výpočet tenzoru hydraulické vodivosti a jeho inverzi, velikost elementu, výpočet středu elementu, ...

4. Metody k nastavení odkazů. Jde o metody sloužící k propojení vrcholů (CNode) se stěnami (CSide).

5. Metody k vytváření lokální matice a globální matice.

5.2.5 CSide

Podobně jako předchozí nese tato třída četné odkazy na související třídy. Dále obsahuje geometrické údaje a část výsledků z globální matice. Stěnou je v tomto programu myšlena součást hranice elementu, jejíž geometrický tvar závisí na tvaru elementu. Konkrétně jde o body v případě čar, úsečky v případě ploch a trojúhelníků v případě čtyřstěnů.

Třída stěny nemá žádný zdrojový soubor a je vytvořena pomocí již načtených informací v elementech a uzlech sítě. Metody této třídy souvisí se základní manipulací s privátními datovými proměnnými jako je nastavení a předání hodnoty proměnné. Dále pak k výpočtu potřebných geometrických vlastností jako je velikost, pozice středu, apod.

5.2.6 CEdge

Podobně jako předchozí třída neodpovídá přímo této třídě žádný vstupní soubor. Jednotlivé hrany jsou generovány podle informací z již hotových stěn. Hranou je v popisovaném programu myšlena množina stěn téhož typu, které zaujímají shodnou pozici v prostoru.

5.2.7 CMaterial

Struktura sloužící k načtení informací ze vstupního souboru. Později jsou materiálové koeficienty přesunuty přímo do elementu.

Jediným možným nastavením datových proměnných této třídy je načtení ze souboru. Ostatní metody pak slouží pouze k získáni konkrétních hodnot těchto proměnných.

5.2.8 CBoundary

Tato třída slouží k načtení a uchování potřebných údajů ke klasifikaci okrajové podmínky. Kromě koeficientů potřebných k popsání podmínky obsahuje tato třída odkaz na stěnu, na které je okrajová podmínka uplatněna a případně na příslušnou okrajovou podmínku transportu.

Jako v předešlém případě není možné konkrétní proměnné přiřadit libovolnou hodnotu. Pro většinu proměnných je jedinou možností, jak změnit jejich hodnotu načtení dat ze souboru. Jinak je přístup k proměnným pouze jednosměrný a je umožněno je pouze číst.

5.2.9 CInitial

Pomocí seznamu těchto tříd jsou načteny počáteční podmínky. Po vytvoření topologie v programu jsou jednotlivé hodnoty počátečních podmínek nahrány do příslušných proměnných daného elementu.

Metody této třídy jsou vesměs určeny pouze pro čtení zadaných hodnot. Hodnoty jsou nastaveny při čtení ze vstupního souboru.

5.2.10 CTransport_bcd

V této třídě je uložen jeden řádek z TRB-souboru (koncentrace látek na hranici studované oblasti). Nastavení těchto hodnot se provádí pouze při načítání dat ze vstupního souboru. Ostatní metody jsou určené pouze pro čtení daných proměnných.

5.2.11 CConcentration

Seznam těchto tříd reprezentuje CON-soubor. Po vyřešení topologie jsou načtené hodnoty přepsány do příslušných proměnných třídy CElement.

Metody této třídy opět slouží pouze k získání hodnot a nastavení jejich hodnot probíhá jen při načítání ze souboru.

5.2.12 CSource

Obsahuje informaci o velikostí vtlačení nebo sání a odkaz na příslušný element, kde se tato informace uplatňuje.

Metody se neliší od předchozích a slouží opět k načtení hodnot ze souboru a dále pak umožňují jednosměrný přístup k datům.

5.2.13 CNeighbour

Seznam tříd CNeighbour je důležitý pro řešení topologie a určení propojení jednotlivých stran a elementů.

Metody třídy lze rozdělit podle jejich použití do několika skupin:

- Metody určené pro čtení proměnných. Slouží k získání přístupů k proměnným. U některých (jde především o nastavené ukazatele na ostatní třídy) je prováděna kontrola, zda daný prvek může existovat, jelikož je uložen v dynamicky alokovaných polích.
- Metody určené k načtení dat ze souboru.
- Metody určené k vytváření sousednosti. Jde spíše o pomocné metody, jelikož o hledání sousednosti se stará třída CMesh, která má lepší přístup k ostatním třídám.
- Metody určené k nastavení vnitřních proměnných. U proměnných, které jsou dynamicky alokované je prováděna kontrola, jestli je nová hodnota vkládána do existující pozice. Tyto metody se především používají při vytváření sousednosti a tvorbě topologie.

5.2.14 CMatrix

Tato třída slouží k uchování globální matice systému, její pravé strany a jejího řešení. Proměnné této třídy jsou předávané řešiči soustavy lineárních rovnic. Matice soustavy je ukládána v komprimované formě nazývané CRS. K uložení je potřeba tří polí. První dvě mají tolik členů, kolik je v matici nenulových prvků. Třetí pole má $n + 1$ hodnot, kde n je počet řádků. První pole slouží k uložení hodnot, Druhé k uložení indexů sloupců ve kterých se vyskytují nenulové prvky. Třetí pole udává začátek nového řádků.

Metody třídy lze rozdělit do tří skupin podle činnosti kterou vykonávají:

1. Metody přístupu k proměnným. Jde jak o nastavení daných hodnot, tak i o jejich čtení.

2. Metody k převodu pro konkrétní řešič. Slouží k převodu do tvaru, který je vyžadován konkrétním řešičem použitým k výpočtu řešení.

3. Metody čtení výsledků z řešiče. Pomocí těchto metod jsou výstupní data z řešiče lineární soustavy převedena do vnitřních proměnných této třídy.

5.2.15 CError

Tato třída slouží k uchování identifikačního čísla chyby, textové zprávy a určení, zda se jedná o programátorskou chybu či uživatelskou. Pole těchto tříd je jednou z proměnných následující třídy a je deklarováno přímo ve zdrojovém kódu.

Na rozdíl od ostatních tříd jsou proměnné této třídy veřejné a tato třída nemá žádné metody k obsluze těchto dat.

5.2.16 CSystem

Tato třída neodpovídá žádné datové struktuře v původním programu a na rozdíl od všech ostatních tříd je deklarována čistě staticky. To umožňuje použít jejich metod bez nutnosti vytvářet instanci třídy. Přístup k jednotlivým metodám se pak děje pomocí vypsání celého jména, dvojicí dvojteček a následně jména metody. Ukázka je uvedena ve výpisu (7).

Obrázek 7: Použití statické metody

```
CSystem::jmeno_metody(argument1, ...)
```

Třída CSystem byla vytvořena jako rozhraní mezi programem a okolím. Stará se o otevírání, zavírání, čtení a zápis do souboru, výpis zpráv a v neposlední řadě i o ukončení programu v případě výskytu chyby.

Nyní se blíže podíváme na jednotlivé metody:

print_usage() Slouží k vypsání návodu na používání. Tato funkce je volána v případě, že nebyly rozpoznány argumenty zadané pomocí příkazové řádky. Po vypsání návodu program čeká na potvrzení a pak se ukončí.

print_info() Slouží k výpisu dat obsažených v této třídě. Tato data byla původně ve struktuře Problem, avšak vzhledem k jejich použití byly přiřazeny k této třídě. Jde především o jméno LOG-souboru a čísla záchytných bodů (argumenty metody checkpoint, viz. níže), které se mají vypisovat.

Tato metoda je volána při výpisu dat třídy CProblem, jelikož data byla původně obsažena ve struktuře, ze které byla odvozena.

checkpoint(int) Tato metoda slouží pro ladící účely. V programu je volána z různých míst s jiným argumentem. Porovnáním hodnoty zadaného argumentu s údaji získanými z inicializačního souboru se zjistí, jestli se má provést výpis vnitřních datových struktur. Pokud daná hodnota záchytných bodů nebyla uvedena v INI-souboru, nic se neprovede a program pokračuje jakoby tato metoda nic nedělala.

Při zjištění shodnosti se spustí výpis všech vnitřních datových veličin do souboru. Každé datové skupině (elementy, stěny, ...) náleží jeden soubor, jehož jméno je generováno automaticky a skládá se z čísla záchytného bodu, celkového čísla po kolikáté je výpis prováděn a zkratky z názvu datové třídy, jejíž seznam je vypisován do daného souboru.

fopen(char *,int) Metoda určena ke zpřístupnění souborů pro čtení nebo zápis.

Přístup k souborům je řešen pomocí proudů. Jde o třídy definované mimo tento program (v knihovně iostream či fstream). Proud je chápán jako posloupnost bajtů. Cílem proudů je zapouzdření převedení dat z disku a klávesnice na obrazovku nebo na disk. V tomto programu jsou pro přístup k souborům používány dvě třídy:

- *ifstream* pro čtení ze souboru

- *ofstream* pro zápis do souboru

Metoda fopen přijímá dva argumenty. Prvním je cesta k souboru a druhým typ otvíraného souboru. Soubor lze otevřít pro:

1. čtení - ukazatel určující pozici v souboru je nastaven na začátek souboru.
2. zápis - pokud soubor již existuje je jeho obsah vymazán. Pokud neexistuje je vytvořen. Vždy je ukazatel na pozici v souboru nastaven na začátek souboru.
3. přidávání - obsah souboru je zachován a ukazatel je umístěn na konec souboru.

Přestože by bylo možné realizovat čtení a zápis do souboru současně (existují instance třídy ifstream i ofstream) není tato možnost v programu nikde využita a proto nebyla realizována.

fclose() Metoda sloužící k uzavření naposled otevřeného souboru. Pokud není žádný soubor otevřen vede volání této metody k chybě a ukončení programu.

myprintf(int,const char*,...) Metoda určená k výpisu hlášení jak na obrazovku tak do LOG-souboru. Metoda může přijímat více argumentů, minimálně však dva. První slouží k určení priority sdělení. Priorita může nabývat hodnot od jedničky do šestky. Při vypisování zpráv se zadaná hodnota porovná s hodnotou získanou z INI-souboru. Pokud je hodnota zprávy menší zpráva je vypsána. Porovnání se provádí jak pro výpis na obrazovku tak pro výpis do souboru.

Druhým argumentem je text zprávy. Text může obsahovat formátovací znaky umožňující vložení hodnoty. Pokud jsou formátovací znaky obsahem zprávy, je nutné jako další argumenty přidat požadované hodnoty.

printf(const char*,...) Slouží k zápisu dat do souboru. Pro použití této metody je nutné mít otevřený soubor pro zápis nebo přidávání (po-

mocí výše zmíněné metody fopen), jinak volání způsobí chybu a ukončí program. Metoda může jako předchozí přijímat více argumentů. První je ukazatel na text zprávy. Počet ostatních je závislý na tvaru textu, konkrétně jestli jsou v textu použity formátovací znaky vyžadující ukazatele na odpovídající hodnoty, které budou převedeny v text a vloženy místo formátovacích znaků.

fgets(char *s, int n) Slouží k načtení jednoho řádku z již otevřeného souboru. Metoda přijímá dva argumenty. První je ukazatel na pole, kam se uloží načtený text. Druhý pak udává maximální velikost pole.

use_system(char*) Metoda sloužící ke spouštění externích řešičů pomocí příkazové řádky. Argumentem metody je textový řetězec, který je předán příkazové řádce.

terminate(char*,int,int,...) Metoda určená k přerušení vykonávání činnosti programu. Nejčastějším důvodem je zjištění chybového stavu. Předávané argumenty jsou textový řetězec, ve kterém je očekáván název souboru, odkud je daná metoda volána, řádek na kterém je volání metody uvedeno a poslední povinný argument udává, o jakou chybu se konkrétně jedná. Podle zadaného čísla se prochází pole instancí CError a při shodě zadaného čísla a identifikačního čísla chyby je vypsána daná chyba. Ostatní argumenty jsou závislé na konkrétním typu chyby.

Výpis je prováděn jak na obrazovku tak do LOG-souboru pokud již byl zadán jeho název. Po vypsání chyby se program ukončí.

generování souboru Na konec se zmiňme o skupině metod sloužící k vygenerování řídícího souboru pro výpočet, který je předán externímu řešiči. Sem patří metody:

1. write_rid_ghp_gi8
2. write_rid_ghp_si2
3. write_gm6_in

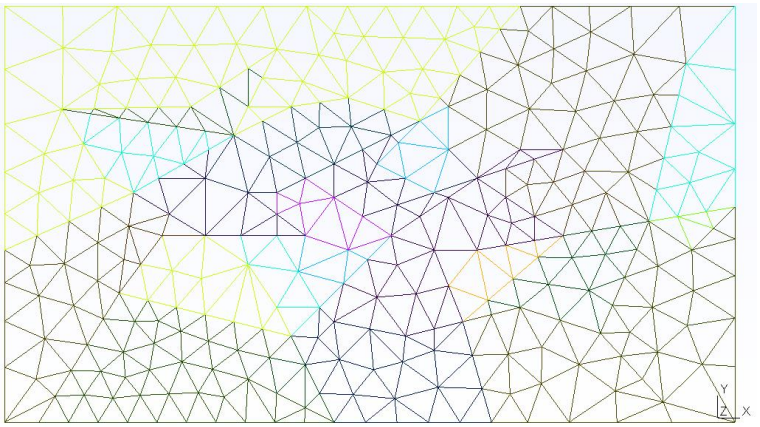
4. write_m_file

Kromě výše uvedených metod jsou v této třídě definovány další, které slouží k jednoduchému nastavení datových položek.

6 Testování činnosti programu

V této kapitole se budeme věnovat srovnání nově vytvořeného programu a jeho předlohy.

Pro testování funkčnosti byly zvoleny dva testovací modely, které jsou součástí přiloženého CD. V prvním případě se jedná o dvojrozměrnou síť znázorněnou na obrázku (8). Tato síť obsahuje 677 elementů a 300 vrcholů. Propojení elementů s odlišnou dimenzí je konformní. 1D elementy mají desetkrát větší hydraulickou vodivost než 2D elementy. Na horním a spodním okraji je zadána homogenní Neumanova podmínka. Na bočních stěnách je zadána Dirichetova podmínka, přičemž na pravé straně je tlaková výška větší.

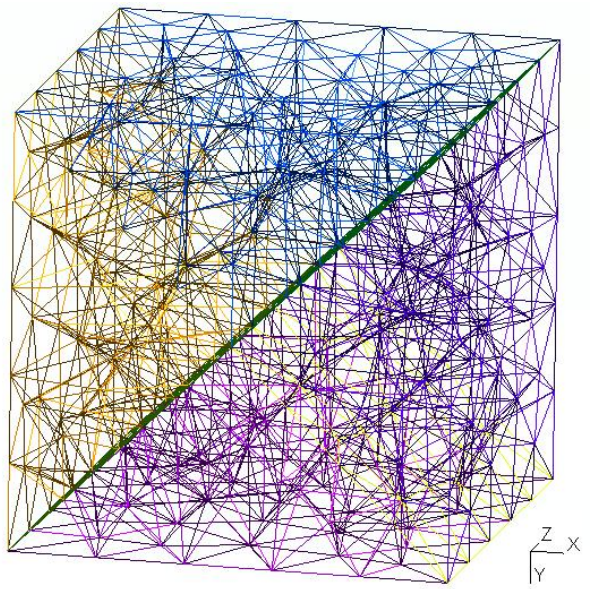


Obrázek 8: Testovací síť pro dvojrozměrný model

Pro druhý model byla zvolena trojrozměrná síť znázorněná na obrázku (9).

Tato síť obsahuje 1 444 elementů a 322 vrcholů. Propojení elementů s odlišnou dimenzí je konformní. Hydraulická vodivost 1D elementů je desetkrát větší než 2D elementů, které jí mají větší než 3D elementy. Okrajová podmínka je na obrázku (11).

Pro oba modely byly vyzkoušeny všechny režimy činnosti programu a porovnány jejich výsledky. Srovnání proběhlo pomocí programu GMSH, kde bylo srovnání provedeno graficky. Výsledky pro dvojrozměrný případ jsou na



Obrázek 9: Testovací síť pro trojrozměrný model

obrázcích (13) a (12).

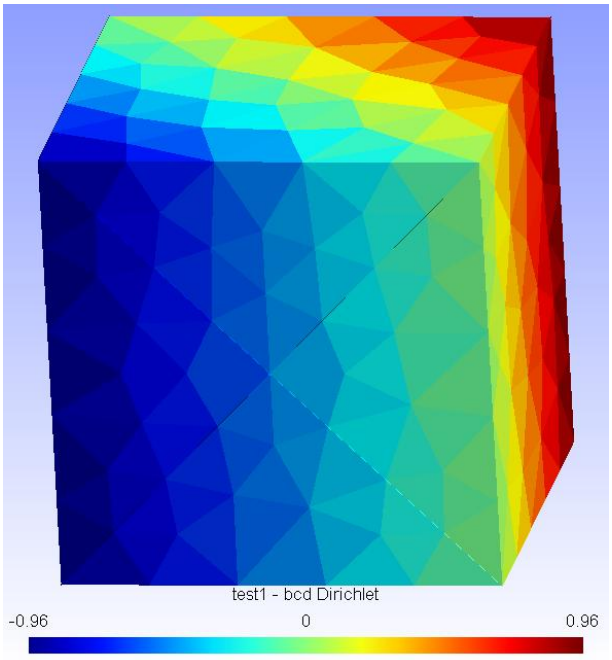
Pro lepší názornost je u třírozměrného modelu znázorněna jen velikost toku (obrázky (15) a (14)).

Druhé srovnání proběhlo pomocí porovnání obsahu souboru. Při tomto porovnání je se zjistily rozdíly. Ve většině případech šlo o jinak zaokrouhlenou poslední číslici.

Obrázek 10: Znázornění rozdílů mezi výsledky

```
1: VP (0.5 , 0.5 , 0.300000000001 ) {0 , 0 , 0.000591382210608 };
2:
3: VP (0.5 , 0.5 , 0.300000000001 ) {0 , 0 , 0.000591382210609 };
```

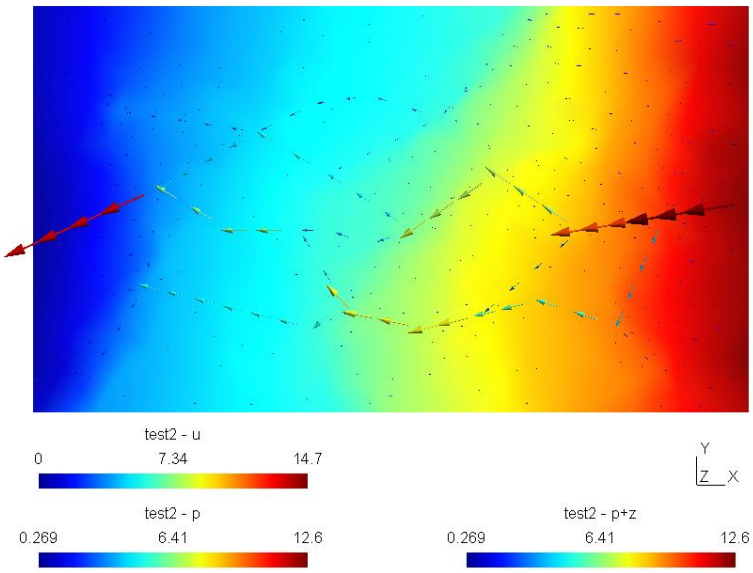
Ve výpisu (10) je znázorněn myšlený rozdíl. Jde o řádky z vypočtených POS souboru pro trojrozměrný model (konkrétně jde o řádek 2 897). Na prvním řádku je uveden hodnota vytvořená novým programem.



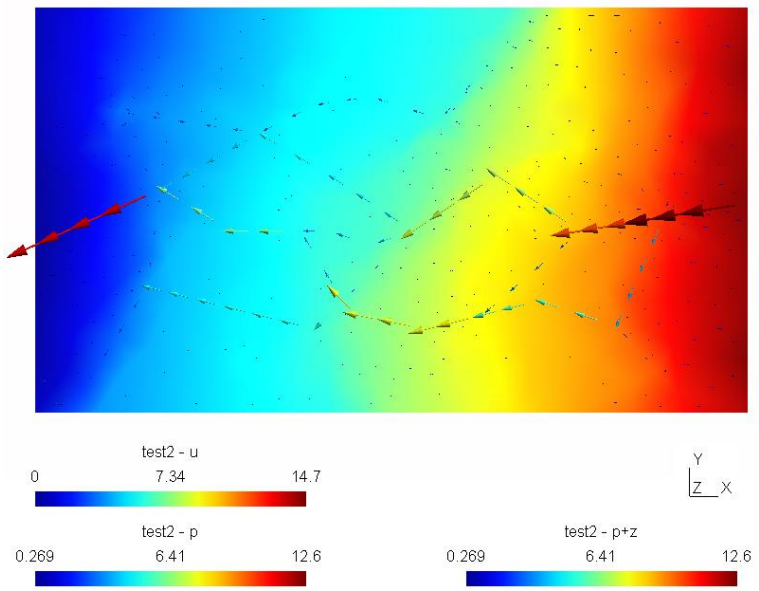
Obrázek 11: Testovací síť pro trojrozměrný model

6.1 Testování doby výpočtu

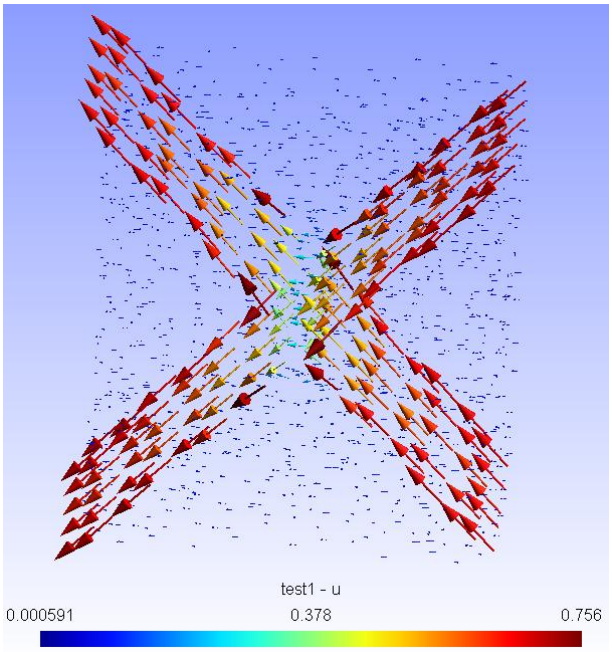
Kromě porovnání výstupů proběhlo i srovnání z hlediska časů výpočtu, kde si však nový program vedl hůře. Při dvacetinásobném spouštění výpočtu v cyklu, který byl realizován pomocným programem, byl naměřen čas jednoho výpočtu čtyři a půl vteřiny pro původní program a pět vteřin pro program nový. Test byl proveden na dvojrozměrném modelu. Při výpočtech byl použit procesor *Core 2 duo* na 2.33Ghz.



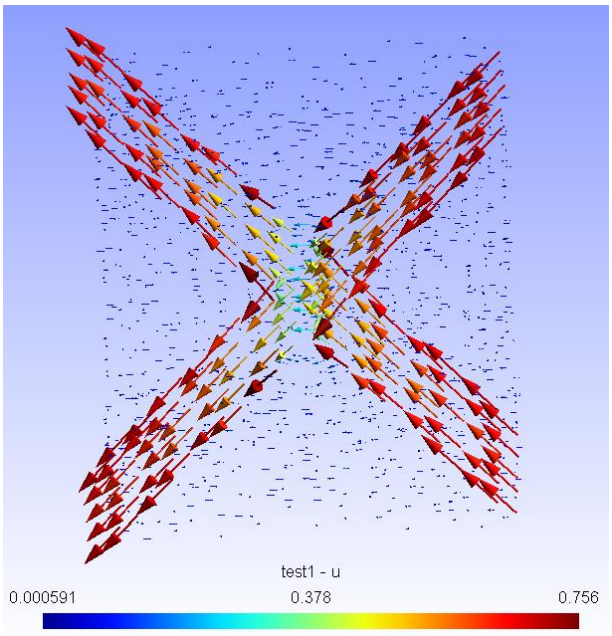
Obrázek 12: Výsledek výpočtu pro původní program 2D modelu



Obrázek 13: Výsledek výpočtu pro nový program 2D modelu



Obrázek 14: Výsledek výpočtu pro původní program 3D modelu



Obrázek 15: Výsledek výpočtu pro nový program 3D modelu

7 Závěr

Na závěr shrňme výsledky práce a nejpodstatnější rozdíly mezi novým a původním programem do několika bodů:

- Nový program je psán objektově.
- Lineární obousměrný seznam je implementován do parametrické třídy. Tím došlo k oddělení způsobu ukládání dat od dat samotných.
- K souborům se přistupuje pomocí proudů.

Při překladu zdrojových kódů obou programů (nového i původního), nebyla nalezena správná knihovna pro připojení interního řešiče a pro většinu externích řešičů nepracoval správně ani původní program. V uvedené verzi pracuje program pouze s externím řešičem Matlab.

Převedením na objektově orientovaný program není vývoj ukončen. Dále je možné se zaměřit:

- Na odstranění problému s řešiči, především pak s připojením interního řešiče nalezením vhodné knihovny nebo změnou překladače.
- Prohloubení objektového přístupu vytvoření bázové třídy obsahující opakující se metody (např. práce s identifikačním číslem).
- Aplikovat dědičnost a polymorfismus na třídy, u kterých mohou jednotlivé instance představovat objekty různých dimenzí.

Reference

[1] Jesse Liberty *Naučte se C++ za 21 dní* ;Computer Press, Praha, 2002

[2] Maryška J., Severýn O., Tauchman M., Tondr D.: Modelling of processes in fractured rock using FEM/FVM on multidimensional domains. Journal of Computational and Applied Mathematics, 215/2, 2008, pp. 495-502. ISSN: 0377-0427

[3] C. Geuzaine, J.-F. Remacle, Gmsh Reference Manual, 2008.

[4] Otto Severý flow123D hp.pdf

[5] Šembera J., Maryška J., Královcová J., Severýn O.: A novel approach to modelling of flow in fractured porous medium, Kybernetika, Vol. 43/4, pp. 577-588, 2007. ISSN 0023-5954

A Příloha

V příloze jsou uvedeny výpisy deklarace jednotlivých tříd vytvořených v programu.

A.1 Výpis třídy CBoundary

```
class CBoundary
{
 private:
// Data readed from bounary conditions files
int         Id; // Id number of condition
int         Type; // Type of boundary condition
double      Scalar; // Scalar - for Dirichlet's or Newton's type
double      Flux; // Flux - for Neumann's type or source
double      Sigma; // Sigma koef. - for Newton's type
int         Where; // Type of location of the condition
int         Nid; // ID number of node where prescribed
int         Eid; // ID number of element where prescribed
int         Sid; // Local # of side, where prescribed
int         Group; // Group of condition
// Topology of the mesh
class CSide         *Side; // side, where prescribed
 class CTransport_bcd *Transport_bcd;  // transport boundary condition
// Matrix
int          F_row; // Row in block F
double       F_val; // Value in block F
double       Rhs; // Value in rhs
// Misc
int          Aux; // Auxiliary flag
double       Faux; // Auxiliary number
 public://-----------------------------------------
               CBoundary();
  int          id() const;
  int          eid() const;
  int          sid() const;
  int          where() const;
```

```
  int           type() const;
  int           f_row() const;
  double        scalar() const;
  double        rhs() const;
  double        f_val() const;
  double        flux() const;
  int           aux() const;
  int           group() const;
  double        sigma() const;
  class CSide*  side()const;
  class CTransport_bcd  *transport_bcd()const;
  void          parse_boundary_line(char *line );
  static void   skip_to_boundaries( std::ifstream *in );
  void          set_side(class CSide*);
  void          set_aux(int);
  void          set_transport_bcd(class CTransport_bcd*);
  void          calc_boundary_f_row();
  void          calc_boundary_f_val();
  void          calc_boundary_rhs();
};
```

A.2 Výpis třídy CConcentration

```
class CConcentration
{
 private:
// Data readed from concentration files
int     Id; // Id number of condition
int   Eid; // ID number of element where prescribed
 static int  N_subst; // # of substances
 double     *Conc;     // Values of concentrations

// Misc
```

```
int       Aux; // Auxiliary flag
double    Faux; // Auxiliary number
 public://-----------------------------------------------
      CConcentration();
 int         id() const;
 int         eid() const;
 double      conc(int) const;
 static void set_n_subst(int n);
void        parse_concentration_line(char *line );
 static void skip_to_concentrations( std::ifstream *in );
};
```

A.3 Výpis třídy CEdge

```
class CEdge
{
 private:
// Basic
int                Id; // Id # of the edge
// Topology of the mesh
int                N_sides; // # of sides of edge
class CSide      **Side; // sides of edge
class CNeighbour *Neigh_vb; // "Compatible" neighbouring
class CNeighbour *Neigh_bb;
// Matrix
int     C_row; // # of row in block C (and E and F) (MH)
double  F_val; // Value in block F
// Misc
int                Aux; // Auxiliary flag
double             Faux; // Auxiliary number
 public://-----------------------------------------------
             CEdge();
 void        set_id(int);
```

```
 void        set_n_sides(int);
 void        set_aux(int);
 int         id() const;
 int         c_row() const;
 double      f_val() const;
 int         n_sides() const;
 void        set_neigh_bb(class CNeighbour*);
 void        set_neigh_vb(class CNeighbour*);
 void        inc_sides(int value=1);
void        make_new_side(int);

void        set_side(int,class CSide*);
void        add_side(class CSide*);
void        set_c_row(int);
void        set_f_val(double);
class CNeighbour   * neigh_vb()const;
class CSide        * side(int)const;
 int           edge_has_vb_neighbour( class CElement **ele,
                                      class CSide **);
};
```

A.4 Výpis třídy CElement

```
class CElement
{
 private:
// Data readed from mesh file
int      Id; // Id # of element
int      Type; //
int     *Nid;         // Id # of nodes of element
int      Mid; // Id # of material
int      Rid;           // Id # of region
 double   Size;          // Area of 1D elm or height of 2D elm
```

```
// Type specific data
int      Dim; // 1 or 2 or 3
int      N_sides; // Number of sides
int      N_nodes; // Number of nodes
// Topology of mesh

 class CNode      **Node; // Element's nodes
 class CMaterial   *Material; // Element's material
class CSide      **Side; // Element's sides

int            N_neighs_vv;// # of neighbours, V-V type (noncomp.)
class CNeighbour **Neigh_vv;// List og neighbours, V-V type (noncomp.)
int            N_neighs_vb;// # of neighbours, V-B type (comp.)
class CNeighbour **Neigh_vb;// List og neighbours, V-B type (comp.)
class CSource    *Source; // Internal source/sink in the element
 class CInitial      *Initial;   // Initial condition
 class CConcentration *Start_conc;// Start concentration  - if prescribed
int         N_subst; // Number of substances
// Material properties
double   K[ 3 ][ 3 ];// Tensor of hydraulic conductivity
double   A[ 3 ][ 3 ];// Tensor of hydraulic resistance (k^(-1))
// Geometrical properties
double      Metrics; // Metrics of the element
double      Centre[ 3 ]; // Centre of mass of element
 // Parameters of the basis functions
 double     *Bas_alfa;      // Parameters alfa
 double     *Bas_beta;      // Parameters beta
 double     *Bas_gama;      // Parameters gama
 double     *Bas_delta;      // Parameters delta

// Matrix
double    **Loc; // Local matrix

double     *Rhs; // Rhs - vector q1, gradients
double      Rhs_b; // Rhs - vector q2, sources
 int         Rhs_b_stat_index;
 double      Rhs_b_stat;
int         A_row; // # of first row in the block A
int         B_row; // # of row in the block B
double     *B_val;    // Values in block B
int         D_row_count; // # of nonzeros in row of the block D
```

```
int        *D_col; // columns with nonzeros in D
double     *D_val; // values of entries in D
 int         D_val_stat_index;
 double      D_val_stat;
int         E_row_count; // # of nonzeros in row of the block E
int        *E_col; // columns with nonzeros in E
double     *E_val; // values of entries in E
// Results
double      Vector[ 3 ]; // Vector quantity - velocity of flow
double      V_length; // Length of vector quantity
double      Scalar; // Scalar quantity (piez. head or pressure)
double      Pscalar; // As scalar but in previous time step
double      Balance; // Balance of flux
 double     *Conc;         // Concentrations in the current time
 double     *Pconc;        // Concentrations in the previous time
// Misc
int         Aux; // Auxiliary flag
double      Faux; // Auxiliary number


 char        supported_element_type( int type );
 void        element_type_specific( );
 void        element_allocation_independent();

 void        side_to_node_triangle(  );
 void        side_to_node_line( );
 void        side_to_node_tetrahedron( );
 void        calc_a_from_k( );
 void        calc_k_from_a( );
 double      element_length_line( );
 double      element_area_triangle( );
 double      element_volume_tetrahedron( );
 void        calc_d_row_count( );
void        alloc_and_init_block_d( );
void        alloc_and_init_block_e( );
void        local_matrix_line( );
void        local_matrix_triangle( );
void        local_matrix_tetrahedron( );
void        node_coordinates_triangle( double[ 3 ][ 2 ] );
void        side_midpoint_triangle( double[ 3 ][ 2 ],
                                    double[ 3 ][ 2 ] );
```

```
void        basis_functions_tetrahedron(double [ 4 ], double [ 4 ],
                                        double [ 4 ], double [ 4 ] );
double      polynom_integral_tetrahedron( double[ 10 ] );
public://-------------------------------------------------
             CElement();
static void  skip_to_elements( std::ifstream *in );

int          id() const;
int          mid() const;
int          rid() const;
int          n_nodes() const;
int          n_neighs_vv() const;
int          n_neighs_vb() const;
int          nid(int) const;
int          aux()const;
int          type()const;
int          dim() const;
int          b_row() const;
int          d_row_count()const;
int          e_row_count()const;
int          a_row()const;
int          d_col(int)const;
int          e_col(int)const;
int          d_val_stat_index( )const;
int          rhs_b_stat_index( )const;

double       centre(int) const;
double       vector(int) const;
double       rhs(int) const;
double       rhs_b() const;
double       scalar() const;
double       size() const;
double       d_val_stat() const;
double       metrics() const;
double       pscalar() const;
double       rhs_b_stat( )const;
double       loc(int,int) const;
double       b_val(int) const;
double       d_val(int) const;
double       e_val(int) const;
double       conc(int) const;
```

```
 double       pconc(int) const;
class CNode*           node(int)const;
 class CSide*           side(int)const;
 class CInitial*        initial()const;
 class CSource*         source()const;
 class CMaterial*       material()const;
 class CConcentration*  start_conc()const;
 class CNeighbour*      neigh_vv(int) const;
 class CNeighbour*      neigh_vb(int) const;
 void                   parse_element_line(char *line );
 int                    n_sides() const;
 void         set_size(double);
 void         set_material(class CMaterial*);
 void         set_initial(class CInitial*);
 void         set_source(class CSource*);
 void         set_start_conc(class CConcentration*);
 void         set_node(int, class CNode*);
 void         set_side(int, class CSide*);
 void         set_aux(int );
 void         set_a_row(int );
 void         set_b_row(int );
 void         set_d_val(int, double );
 void         set_conc(int, double );
 void         set_d_val_stat(double );
 void         set_rhs_b(double );
 void         set_scalar(double );
 void         set_balance(double );
 void         inc_balance(double );
 void         set_d_val_stat_index(int );
 void         set_rhs_b_stat_index(int );
 void         set_neigh_vv(int,class CNeighbour*);
 void         set_neigh_vb(int,class CNeighbour*);
 void         init_n_neighs_vv();
 void         init_n_neighs_vb();
 void         inc_aux();
 void         inc_rhs(int,double=1);
 void  side_to_node();
 void  make_pscalar();
 void  calc_material_tensors(  );
 void  calc_metrics(  );
 void  calc_centre(  );
```

```
 void  calc_rhs(  );
 void  calc_b_val(  );
 void  dirichlet_elm(  );
 void  calc_rhs_b( int );
 void  make_block_d(int);
 void  make_block_e( );
 void  elm_alloc_concentrations(int);
 void  local_matrix( );
 void  make_element_vector_line();
 void  make_element_vector_triangle();
 void  make_element_vector_tetrahedron();
 void  init_v_length();
 void  scalar_to_pscalar();
 void  conc_to_pconc(int);
 void  transport_same_dim( double *, int *, int ,double );
 void  transport_comp_model( double *, int *, int , double );
 void  transport_non_comp_model( double *, int *, int, double );
 int   elements_are_vb_neighbours(class CElement *);
};
```

A.5 Výpis třídy CError_message

```
class CError_message
{
 public:
    int   Rc;             // Return code of the program
    char *Message;        // Message describing the error
    int   Type;           // PROGRAMMER'S / USER'S
};
```

A.6 Výpis třídy CInitial

```
class CInitial
{
 private:
// Data readed from initial conditions files
 int    Id; // Id number of condition
 int    Eid; // Id number of element where prescribed
 double Epress; // Press in the element's center
 int    N_sides; // # of sides of the element readed from file
 double *Spress; // Presses on the sides of the element
 double *Sflux; // Fluxes via sides of the element
 int     N_neighs_vv; // # of neighbours, V-V type (noncomp.)
 int   *Hid; // ID numbers of neighbour
// Topology of the mesh
// Misc
int       Aux; // Auxiliary flag
double    Faux; // Auxiliary number
 public://----------------------------------------
              CInitial();
  int         id() const;
  int         eid() const;
  int         n_sides() const;
  int         n_neighs_vv() const;
  double      epress() const;
  void        parse_initial_line(char *line );
  static void skip_to_flow_field( std::ifstream *in );
};
```

A.7 Výpis třídy CMaterial

```
class CMaterial
{
```

```
 private:
// Data readed from material file
int      Id; // Id # of the material
int      Type; // Type of material
int      N_coefs; // # of coefficients
double  *Coef; // Values of coefficients
// Misc
int      Aux; // Auxiliary flag
double   Faux; // Auxiliary number
void material_type_specific( );
char supported_material_type( );
 public://-----------------------------------------
               CMaterial();
   int         id() const;
   int         type() const;
   double      coef(int) const;
  void        parse_material_line(char *line );
  static void skip_to_material( std::ifstream *in );
};
```

A.8 Výpis třídy CMatrix

```
class CMatrix
{
 private:
int       Id; // Id # of the matrix
int       Size; // Size of the matrix
int       N_nonzeros; // # of nonzeros in matrix
int      *I; // i-vector for CSR storage
int      *J; // j-vector for CSR storage
```

```
double   *A; // a-vector for CSR storage
double   *Rhs; // Right hand side vector
double   *Solution; // Vector of solution
int       SizeA; // Size of the A-block
int       SizeB; // Size of the B-block
int       SizeC; // Size of the C-block
int       SizeC0; // Size of the C*-block
 void insert_value( int , int , double );

 public:
           CMatrix();
  int      id() const;
 void      set_id( int);
 void      set_sizeA( int);
 void      set_sizeB( int);
 void      set_sizeC( int);
 void      set_sizeC0( int);
 void      set_size(int);
 void      init_size();
 void      set_n_nonzeros(int);
 void      alocate_vectors( );
 void      init_vectors( );
 void      make_i_vect(CList<CElement> & ,int);
 void      make_j_vect(CList<CElement> & ,int ,int ,int  );
 void      make_a_vect(  CList<CElement> &, CList<CBoundary> &,
                         CList<CEdge> & );
 void      make_rhs_vect(  CList<CElement> &, CList<CBoundary> &,int  );
 int       n_nonzeros()const;
 int       size()const;
 int       sizeA()const;
 int       sizeB()const;
 double    solution(int)const;
 void      set_a(int,double);
 void      set_rhs(int,double);
 void      write_vector_sro( )const;
 void      write_matlab_matrix( )const;
 void      write_matlab_rhs( )const;
 void      read_vector_sri( );
 void      read_gm6_out( );
 void      read_matlab_solution( );
 void      make_whole_matrix( struct CMatrix &);
```

```
 void      inc_I(int=1);
 void      inc_J(int=1);
 void      deallocate_vectors();
 int       *i()const;
 int       *j()const;
 double    *a()const;
 double    *rhs()const;
 double    *solution()const;
};
```

A.9 Výpis třídy CMesh

```
class CMesh
{
 private:
int              Id;                  // Id number of the mesh
// Files
   char            *Geometry_fname;       // Name of file of nodes and elems
   char           *Material_fname;       // Name of file of material coeffs.
   char            *Boundary_fname;       // Name of file of bound. cond.
   char            *Initial_fname;        // Name of file of initial cond.
   char            *Neighbours_fname;     // Name of file of topology
   char            *Sources_fname;        // Name of file of sources
   char            *Concentration_fname;  // Name of file of concentration
   char            *Transport_bcd_fname;  // Name of file of transport BCD
// Lists
   int                     N_nodes;   // # of nodes
   CList<class CNode>       Node;        // list of nodes
   int                      N_elements;  // # of elements
   CList<class CElement>    Element;      // list of elements
   int                      N_materials;  // # of materials
   CList<class CMaterial>   Material;    // list of material
   int                    N_boundaries; // # of boundary conditions
```

```
   CList<class CBoundary>   Boundary;   // list of  boundary condition
   int                   N_initials; // # of initial conditions
   CList<class CInitial>    Initial;   // list of  initial condition
   int             N_concentrations;// # of concentrations
   int           N_substances;    // # of substances transported by water
   CList<class CConcentration> Concentration;// list of concentration
   int                      N_transport_bcd;
   CList<class CTransport_bcd> Transport_bcd;
   int                      N_sources;       // # of sources
   CList<class CSource>        Source;          // list of source
int                    N_sides;         // # of sides
int                    N_insides;       // # of internal sides
int                    N_exsides;       // # of external sides
CList<class CSide>      Side;            // list of side

int                     N_edges;         // # of edges
CList<class CEdge>      Edge;            // list of edge

int                     N_neighs;        // # of neighbours
CList<class CNeighbour>  Neighbour;       // list of neighbour



 int              count_sides();
 int              count_edges();
 void             parse_element_properties_line( char *line );
 void             set_element_property( int id,
                                        int type, double value);
 int              skip_to_element_properties( std::ifstream * );
 void             neigh_bb_to_element( class CNeighbour *ngh );
 void             calc_a_row( );
void             calc_b_row( );
 double           calc_water_balance( int c_water );
 void             transport_node_conc( double *pi, int sbi);
 void             make_neighbours_bb( );
 void             make_neighbours_vb(  );
 void             make_neighbours_vv(  );
 public://-------------------------------------------------
  CMesh();
 ~CMesh();
```

```
 void set_id(int);
 int  id() const;
 int  n_sides() const;
 int  n_elements() const;
 int  n_edges() const;
 int  n_exsides() const;
 int  n_substances() const;
 bool source_exist() const;
 bool concentration_exist() const;

 void set_geometry_fname(char *);
 void set_material_fname(char *);
 void set_boundary_fname(char *);
 void set_initial_fname(char *);
 void set_concentration_fname(char *);
 void set_transport_bcd_fname(char *);
 void set_neighbours_fname(char *);
 void set_sources_fname(char *);
 void set_n_substances(int);

 void read_node_list( );
 void read_element_list();
 void read_material_list();
 void read_boundary_list();
 void read_initial_list();
 void read_concentration_list();
 void read_transport_bcd_list();
 void read_neighbour_list();
 void read_source_list();
 void read_element_properties();

 void make_side_list();
 void make_edge_list();

 void create_node_hash( );
void create_element_hash( );
void create_material_hash( );
void create_boundary_hash( );
 void create_neighbour_hash( );
void create_source_hash( );
 void create_concentration_hash( );
```

```
 void create_transport_bcd_hash( );

 void element_to_material( );
 void element_to_node( );
 void node_to_element(  );
 void element_to_side_both(  );
 void neigh_vv_to_element( );
 void element_to_neigh_vv( );
 void neigh_vb_to_element_and_side( );
 void neigh_bv_to_side( );
 void element_to_neigh_vb( );
 void side_shape_specific( );
 void side_to_node( );
 void node_to_side(  );
 void neigh_bb_topology( );
 void neigh_bb_to_edge_both( );
 void edge_to_side_both(  );
 void neigh_vb_to_edge_both( );
 void side_types( );
 void count_side_types( );
 void boundary_to_side_both( );
 void source_to_element_both( );
 void initial_to_element( );
 void concentration_to_element( );
 void transport_bcd_to_boundary( );
 void  make_ele_initials();

void  output_convert_to_pos_source(int);
void  output_convert_to_pos_bcd(int,char*);
void  output_convert_to_pos_material(int);
void  output_convert_to_pos_concentration(int);
void  output_convert_to_pos_transport_bcd(int,char*);
void  output_flow_field_in_time(int);
   void  output_transport_init(int);
   void  output_transport_time(int);
   void  output_convert_to_pos_scalar(int out_digit,
                                      char *description, bool );
   void  output_convert_to_pos_vector(int out_digit,
                                      char *description );

void  edge_calculation_mh();
```

```
void  element_calculation_mh(int,int);
void  side_calculation_mh(  );
void  boundary_calculation_mh( );
void  local_matrices_mh( );
int   calc_nonzeros( );
 int   calc_nonzeros_for_gm6();

 CList<CElement>& elements();
 CList<CBoundary>& boundaries();
 CList<CEdge>& edges();

 void  calculation_unsteady( class CMatrix *matrix,double);
 void  make_side_flux(class CMatrix* );
 void  make_element_scalar(class CMatrix * );
 void  make_element_vector( );
 void  make_sides_scalar( class CMatrix *);
 void  make_node_scalar( );
 void  make_neighbour_flux( );
 void  make_previous_scalar();
 void  calc_element_balance( );
 void  water_balance( );
 double transport_time_step();
 double *transport_aloc_mi();
 double *transport_aloc_pi();
 void    transport_init( double *pi);
 void    transport( double*, double*,double);

 CHelp_node*  init_help_node();
 void         read_data_to_help_node(CHelp_node*);
 void         output_transport_convert(CHelp_node*,
                                       int ,int ,  char* );
 void         make_edge_list_ngh( );
 void         make_neighbours( );
 void         write_neighbours();
};
```

A.10 Výpis třídy CNeighbour

```
class CNeighbour
{
 private:
 int     Id;         // Global id number
int     Type;       // Type
int     N_sides; // # of neighbouring sides
int     N_elements; // # of neighbouring elements
int     N_coefs; // # of coefficients
int    *Sid; // id numbers of neighbouring sides
int    *Eid; // id numbers of neighbouring elements
 double  Flux;    // flux for VV neigh. from e1 into e2
                  // is negative, from e2 into e1 is positive
double          *Coef; // coefficients of neighbouring
class CEdge     *Edge; // edge (set of neighbouring sides)
class CSide    **Side; // neighbouring sides
class CElement **Element;  // neighbouring elements
char            *Line;

// Misc
int     Aux; // Auxiliary flag
double  Faux; // Auxiliary number

char supported_neighbour_type( int  );
void neighbour_specs_bb_e();
void neighbour_specs_bb_el();
void neighbour_specs_vb_es();
void neighbour_specs_vv_2e();

 int elements_common_sides(class CElement *e0,
                           class CElement *e1, int s[ 2 ] );
 int elements_common_sides_1D(class CElement *e0,
                              class CElement *e1, int s[ 2 ] );
 int elements_common_sides_2D(class CElement *e0,
                              class CElement *e1, int s[ 2 ] );
 int elements_common_sides_3D(class CElement *e0,
                              class CElement *e1, int s[ 2 ] );
```

```
 public://-----------------------------------------------
                   CNeighbour();
  int              id() const;
  int              sid(int) const;
  int              eid(int) const;
  class CElement*  element(int) const;
  class CSide*     side(int) const;
  class CEdge*     edge() const;
  double           coef(int);
  int              type() const;
  int              n_elements() const;
  int              n_sides() const;

  static void      skip_to_neighbours( std::ifstream *in );
  void             parse_neighbour_line(char *line);
  void             neighbour_type_specific();
  void             make_elements(int);
  void             make_sides(int);
  void             make_coefs(int);

  void             set_id(int);
  void             set_type(int);
  void             set_eid(int,int);
  void             set_sid(int,int);
  void             set_element(int,class CElement*);
  void             set_side(int,class CSide*);
  void             set_coef(int,double);
  void             set_edge(class CEdge*);
  void             set_flux(double);

  void             sort_eid();
  void             write_output();
  void             neigh_bb_el_to_side( );
  void             neigh_bb_e_to_side( );
};
```

A.11 Výpis třídy CNode

```
class CNode
{
 private:
  // Data readed from mesh file
  int       Id;               // Id number of node
  double    X;               // X coordinate of node
  double    Y;               // Y coordinate of node
  double    Z;               // Z coordinate of node
  // Topology
  int            N_elements; // # of elms connected by the node
  class CElement **Element;    // List of elements
  int            N_sides;   // # of sides connected by the node
  class CSide    **Side;      // List of sides
  // Results
  double   Scalar;   // Scalar quantity (pressure/piez. head)
  double   Pscalar;   // As scalar but in previous time step
  int      N_subst;    // Number of substances
   double     *Conc;      // Concentration in the current time
  // Misc
  int      Aux;        // Auxiliary flag
  double   Faux;       // Auxiliary number

 public://------------------------------------------------------------
              CNode();
             ~CNode();
   int         id() const;
   double      x() const;
   double      y() const;
   double      z() const;
   static void skip_to_nodes( std::ifstream *in );
  void        parse_node_line(char *line );
  int                   n_elements() const;
  int                   n_sides() const;
  double                scalar() const;
  double                faux() const;
  double                conc(int) const;
```

```
  class CElement*       element(int)const;
  class CSide*          side(int)const;
  void                  set_n_elements(int);
  void                  set_scalar(double);
  void                  set_faux(double);
  void                  inc_scalar(double);
  void                  inc_faux(double);
  void                  inc_n_elements();
  void                  set_n_sides(int);
  void                  set_conc(int,double);
  void                  inc_conc(int,double);
  void                  inc_n_sides();
  void                  init_sides();
  void                  add_side(class CSide*);
  void                  init_element();
  void                  add_element(class CElement*);
     void                  make_conc(int);
};
```

A.12 Výpis třídy CProblem

```
class CProblem
{
 private:
// Global
 char            *Ini_fname;       // Name of ini file
 int              Type;           // Type of problem
 char            *Description;     // Short description of problem
  double           Stop_time;       // Number of time steps
  double           Save_step;       // Step for outputing results
  double           Time_step;       // Time step for computation
// Filenames
  int             File_type;       // Type of inpit files
```

```
 char            *Mesh_fname;      // Name of file describing mesh
 char           *Material_fname;  // Name of file of material coeffs.
 char            *Boundary_fname;  // Name of file of bound. cond.
  char            *Initial_fname;   // Name of file of initial cond.
  char            *Neighbours_fname;// Name of file describing topology
 char            *Sources_fname;   // Name of file describing sources
 // Transport
  int              Transport_on;    // Compute transport YES/NO
  int       N_substances;    // # substances transported by water
  char      *Concentration_fname;// Name of file of concentration
  char      *Transport_bcd_fname;// Name of file of transport bcd
  char      *Transport_out_fname;// Name of file of trans. output
  char     **Substance_name; // Names of substances
// Mesh
 CMesh            Mesh;            // Mesh for problem
// Constants
 double          G;              // Gravity acceleration
 double          Rho;            // Density of fluid
 // Parametrs
  int              Goal;            // What will flow solve
// Run

// Solver
  char            Manual_run;      // Run solver manualy ?
 char            Use_ctrl_file;   // Own control file ?
 char           *Ctrl_file;       // Name of control file
 char           *Solver_name;     // Name of the solver
 char            *Solver_params;   // Solver's comamnd line parameters
 double           Eps_solver;      // Accracy of solver
 char             Keep_sfiles;     // Keep or remove solver files?
 class CMatrix    Matrix;          // Global matrix of the system
 int             Solver_id;       // Id number of the solver
// Output
 char             Write_output;    // Output YES/NO?
 char            *Out_fname;       // Name of output file
 int             Out_digit;       // # of digits of output
    int              Out_file_type;   // Type of output file

   void  read_ini_file(char* fname);
   bool  check_ini_values();
   void  convert_to_pos();
```

```
  void  compute_mh();
  void  compute_mh_unsteady_saturated();
  void  compute_mh_steady_saturated();
  void  compute_neighbours();
  void  subst_names(char *);
  void  make_hashes();

  void  output();
  void  output_convert_to_pos();
  void  output_convert_to_pos_source();
  void  output_convert_to_pos_bcd();
  void  output_convert_to_pos_material();
  void  output_convert_to_pos_concentration();
  void  output_convert_to_pos_transport_bcd();
  void  output_transport_init();
  void  output_flow_field_init();
  void  output_transport_time(double);

  void  calculation_mh( );
  void  solve_system( );
  void  postprocess( );
  void  output_flow_field_in_time(double );
  void  make_matrix_mh( );
  void  solver_type( );
  void  write_control_file();
  void  write_matrix();
  void  write_rhs( );
  void  run_solver();
  void  run_gm6_int();
  void  read_solution( );
  void  transport( );
  void  output_compute_mh();
  void  output_transport_convert();
  void  write_neighbours();

public://-----------------------------------------------
      CProblem();    // constructor
     ~CProblem();    // destruktor

int   get_params( int argc, char **argv);
void  make(char *fname);
```

```
 void  make_mesh();
 void  topology();
 void  preprocess();
 void  process();

 void  print_all();
 void  print(char *);
 void  print_mesh(char *);
 void  print_nodes(char *);
 void  print_elements(char *);
 void  print_materials(char *);
 void  print_boundaries(char *);
 void  print_neighbours(char *);
 void  print_sides(char *);
 void  print_edges(char *);
 void  print_hashes(char *);
 void  print_matrix(char *);
 void  print_sources(char *);
};
```

A.13 Výpis třídy CSide

```
class CSide
{
 private:
int               Id; // Id # of side
int               Type; // INTERNAL | EXTERNAL
int               Shape; // POINT, LINE, TRIANGLE
int               Dim;
// Topology of the mesh
class CElement   *Element; // Pointer to element to which belonged
int               Lnum; // Local # of side in element
int               N_nodes; // # of nodes
```

```
class CNode    **Node; // Pointers to sides's nodes
class CBoundary *Cond; // Boundary condition  - if prescribed
class CEdge     *Edge; // Edge to wich belonged
class CNeighbour *Neigh_bv; // Neighbour, B-V type (comp.)
// Geometry
double            Metrics; // Length (area) of the side
double            Normal[ 3 ]; // Vector of (generalized) normal
double            Centre[ 3 ]; // Centre of side
// Matrix
int               C_row; // # of row in block C
double            C_val; // Value in block C
// Results
double  Flux; // Flux through side
double  Scalar; // Scalar quantity (piez. head or pressure)
double  Pscalar; // As scalar but in previous time step
// Misc
int           Aux; // Auxiliary flag
double      Faux; // Auxiliary number

double        side_length_line( );
double        side_area_triangle( );
void          side_normal_triangle( );
void          side_normal_line( );
void          side_normal_point( );
void          side_centre_point( );
void          side_centre_line( );
void          side_centre_triangle( );
   int           number_of_common_nodes_ss( class CSide * );
   int           number_of_common_nodes_se( class CElement * );
 public://-----------------------------------------------
              CSide();
  void        set_id(int);
  void        set_type(int);
  void        set_aux(int);

  int         id() const;
  int         aux() const;
  int         lnum() const;
  int         c_row() const;
  double      c_val() const;
  double      metrics()const;
```

```
  double      scalar()const;
  double      flux()const;
  double      normal(int)const;
  double      centre(int)const;
  void        set_flux(double);
  void        set_scalar(double);
 void        set_element(class CElement*);
 void        set_lnum(int);
 void        set_neigh_bv(class CNeighbour*);
 void        make_dim(int);
 void        set_node(int,class CNode*);
 void        set_edge(class CEdge*);
  void        set_cond(class CBoundary*);
 int         n_nodes()const;
 int         type()const;
 int              shape()const;
 class CNode*     node(int) const;
 class CElement*  element() const;
 class CBoundary*  cond() const;
 class CEdge* edge() const;
  class CNeighbour *neigh_bv() const;
  void        calc_side_c_row( );
  void        calc_side_metrics( );
  void        calc_side_normal( );
  void        calc_side_centre( );
  void        calc_side_c_val( );
  void        calc_side_rhs( );
  int         neighbour_sides( class CSide *s1 );
};
```

A.14 Výpis třídy CSource

```
class CSource
```

```
{
 private:
  int             Id;
 int             Type;
 int             Eid;
 double          Density;
 class CElement *Element;
 int             Aux;
 double          Faux;
 public://-----------------------------------------------
             CSource();
 int         id() const;
 int         eid() const;
 double      density() const;
void        parse_source_line(char * );
 static void skip_to_sources( std::ifstream * );
 void        set_element(class CElement*);
};
```

A.15 Výpis třídy CSystem

```
class CSystem
{
 private:
  static  CError_message  Error_message_list[];
  static  int            N_checkpoints;
  static  int           *Checkpoint_list;
  static  int           *Chp_count;
  static  int            Pause_after_run;
  static char     * Log_name;    // Name of the log file
  static int        Log_verb;    // Verbosity of outputs to logfile
  static char       Log_init;    // Is log file initialized?
  static int        Scr_verb;    // Verbosity of outputs to logfile
```

```
 static CProblem * G_problem;
 static char     *   File_name;   // Name of the open file
 static unsigned int File_open;   // Type of the open file
 static std::ofstream  Fout;
 static std::ifstream  Fin;


public:
 static void Problem(CProblem*);
// log file function
 static void ini_log_file(char *);
 static void   set_log_verb(int);
 static void   set_scr_verb(int);
 static void   check_verb();
 static void   set_pause_after_run(int);
 static int    fopen( char *,int );
 static int    fclose();
 static int    fgets( char *s, int n);
 static int    printf(char *, ... );
 static std::ifstream *in_file() ;
 static  void  terminate(char*,int,int,...);
 static  int   myprintf(int,const char*,...);
 static  void  print_usage();
 static void   print_info();
 static  void  checkpoint( int );
 static  int   checkpoint_on( int );
 static  int   checkpoint_count( int );
 static void   set_checkpoints( char * );
 static  int   get_int( const char *, char *, int , char * );
 static  char *get_string( const char *, char *, char *, char * );
 static  char  get_bool( const char *, char *, char , char * );
 static double read_double( );
 static char  *xstrcpy(char * scr);
 static char  *xgetcwd( void );
 static int    xremove( const char * );
 static char  *xstrtok( char *, const char * );
 static int    xchdir( const char *s );
 static void   clean_directory(  );
 static int    use_system(char*);
 static void   write_rid_ghp_gi8( double );
 static void   write_rid_ghp_si2( double );
```

```
  static void   write_gm6_in( double );
  static void   write_m_file( );
  static char*  log_fname();
};
```

A.16 Výpis třídy CTransport_bcd

```
class CTransport_bcd
{
 private:
// Data readed from transport bounary conditions files
int       Id;
int     Bid;
 static int N_subst;
 double   *Conc;
// Misc
int       Aux; // Auxiliary flag
double    Faux; // Auxiliary number
 public://-----------------------------------------------
             CTransport_bcd();
 int         id() const;
 int         bid() const;
 double      conc(int) const;
 static void set_n_subst(int n);
void        parse_transport_bcd_line(char *line );
 static void skip_to_transport_bcd( std::ifstream *in );
};
```
